Glenn McCrea

2508 LaunchCode – Foundations

Practice: Basics of Coding and Computers - 2 - Hello World

"Hello World" in C++

C++, being a compiled language, must be complied or "translated" into machine code prior to being run on a computer system directly. This is a rudimentary summation of the process.

For creating a type of "Hello World," one must first begin by telling the program what files to include this is accomplished by using "#include <filename>." This pre-processor directive, "header files" is used to pull in the file that we notated. In the example I found, the file was "<iostream>." This allows the program to use another command "cout" later in the process to print to the screen.

Following the pre-processor directive, the code begins with a "int main() {…}". One must include the curly braces to indicate the start and the end of the "function" to be processed. The code is then executed from the function " std::cout << "Hello World!"; " which prints the text in quotations and then ends the the statement with the semi-colon. This is considered the output statement.

Finally, to finish the simple program, a statement of "return 0;" is used to indicate the end of the program - the "exit status" - in this case indicating success. From this point, the program is completed and should have output "Hello World!"

Now, from the functional side of things, this is still not code that is machine readable – this is human readable code. Machine readable code is 1's and 0's called binary. To convert human readable code into machine readable code a translator is needed. That is where compiler programs come in, a compiler will read the the human code into machine readable code so that the CPU knows what to do. The compiler produces an executable file that can then be run efficiently and with good performance. The compiler translates the human code entirely before the program is run.

Because the entire thing is processed before it is run, that means that finding errors is slower. One must write the code, compile the code, and then do error checking and code corrections. This comes with the trade off of slower development, but faster operation of working code. Once it is compiled, it runs fast with no intermediary.

Each set of code has its purpose: Human readable code is just that, readable by humans. Machine code, brings it to life in the CPU.

"Hello World" in Python

Contrastingly, using an interpreted language like python, the process looks different. To create a "Hello World" in Python, one simply uses the built in function "print()". Functions are commands that take inputs, applies some rules to the input, and then delivers a result. In this case if we add "print('Hello World') we end up with a message displaying that says 'Hello World'. Python has a lot of built in functions.

With regard to executing the program, the commands must be saved to a ".py" file and then executed in the terminal of VSCode (or another terminal) or with Python IDLE. At this point the human readable code is executed as human readable code in one of terminals through the process of an interpreter.

An interpreter runs the code line by line translating it from human readable code into machine code at the time of execution. Because it is being translated as it is being read, this generally allows for faster debugging because the interpreter will show the errors as they are being interpreted, rather than having to compile the code first.

All in all, one might ask, "if interpreted code is faster to debug, why would it not be the only type of language that is used now?" Operational speed is the answer. Because the translation happens line by line at the time of executing the code, interpreted languages do not operate as quickly as a compiled code does. There is that intermediary process of translation that happens using some of the CPU power and time.

Both types of language have their place. Use cases for interpreted languages would include rapid development, prototyping, and also for applications that are changing frequently. It is good for these things because of the quicker debugging that can happen. Use cases for compiled languages would include performance intensive applications like processing real-time audio where efficiency and performance are crucial.